FLARUM

目錄

Flarum 文档

本文档翻译自 Flarum 官方文档。

Flarum 是一款现代、优雅、简洁、强大的论坛软件。Flarum 让在线交流变得更加轻松愉快。

开始阅读

本手册由 Flarum 中文社区翻译。

Github

本文档源码托管在：https://github.com/justjavac/flarum-doc

目录及翻译者

  - 搜索
- 翻译Flarum
  - 介绍
  - 打包
  - 本地化 @justjavac
  - 升级
  - 分发

> 本文档最后由 @ttnl 维护于 2017 年 1 月 22 日。

前言

介绍

Flarum 是一款专注于简洁的，自由、开源的论坛软件。通过 Flarum 您可以轻松地为您的网站搭建一个论坛。

Flarum 在 GitHub 上开放地开发。

目标

Flarum 是 esoTalk 和 FluxBB 的共同继承者。 它的目标是:

- 快速、简单。 没有混乱，没有膨胀，没有复杂的依赖关系。Flarum 使用 PHP 构建，因此它很容易部署。界面使用 Mithril，它是一个高性能 JavaScript 框架。
- 漂亮、响应式。 Flarum 旨在以人为本，它有着精心的跨平台的、开箱即用的设计。界面布局使用了 LESS，所以主题风格只是小事一桩。
- 强大、可扩展。 为了满足您的社区需求，您可以定制、扩展和集成 Flarum。Flarum 的架构非常灵活，同时提供有全面的 API 和文档。
- 自由、开放。 Flarum 基于 MIT license 发布。

安装

> **Flarum** 目前处于测试阶段，因此不要将它用在生产环境中。 Flarum 的源码在 GitHub。查看开发进度的路线图。

想立即使用 Flarum，可以到下载页面。你需要一个安装了 **PHP 5.5+** 和 **MySQL 5.5+** 的服务器。

核心团队

- Toby Zerner (GitHub, Twitter) —— 核心开发者
- Franz Liedke (GitHub, Twitter) —— 核心开发者
- Daniël Klabbers (GitHub, Twitter) —— 项目管理
- Jordanjay29 (GitHub) - 社区管理
- Dominion (GitHub) – 国际化
- Kulga (GitHub) - 社区主持人
- Digital (GitHub) - 社区主持人

> 译者：@justjavac Last updated by @ttnl 17/1/20

贡献

报告 Bug

感谢您帮助我们测试 Flarum。我们很高兴您能加入我们的团队！我们需要能够耐心地对问题进行检查且能清晰地对其进行沟通的人。也许你已经知道，优秀的 Bug 报告花费时间和精力。如果你没有意见，那么我们就开始吧！

已经发现了 Bug ？太棒了！我们愿闻其详 —— 不过你首先应该检查你是否把时间花费在了已经知道的问题上：

- 搜索我们的支持论坛 查看

请记住，漏洞报告的目标是使我们能轻松地重现此漏洞，并开发出对应补丁。因此，你应该提供尽可能多的相关信息，并且用清晰且基于事实的语言与我们沟通。这篇文章是关于如何有效报告漏洞的绝佳材料。

如需报告漏洞，请在 Github 上的 flarum/core 库中创建问题（issues）。在提交 issues 之前，请搜索一下之前的 issues。

贡献代码

请参阅贡献代码。

> 译者：@ttnl
> Last updated by @ttnl 17/1/20

FAQ

什么时候才能发布 Flarum 的稳定版？

我们也不能保证什么时候可以发布——因为中间的变数太多！不过，你可以按照 Roadmap 关注我们的开发进度。

届时我能把论坛从 Beta 版升级到稳定版么？

可以。然而升级过程可能会略微地“弄脏你的手”。我们会尽量做到使这一过程足够简单。

那么我能不能在实际中用 Beta 版呢？

安装指示中反对这种做法是有原因的。迄今 Flarum 都还不够成熟，亦存在着很大的出错的空间。你使用的扩展可能和更新版的 beta 发生渲染冲突。这些 Bug 可能造成内容丢失或者是大量的用户隐私资料曝光。未达最佳标准的 SEO 可能损害你网站的搜索引擎排名，使得你的网站更难吸引新的用户。现在的用户可能对性能问题或未完成的功能表示不满，甚至离开你的网站。缺点不一而足。

你也可以决定忽略这些风险并安装 Flaurm，这毕竟取决于你 …… 但是你需要注意如果一旦出现问题，处理它们将会是你的责任。我们不能保证能帮助你解决这些问题。所以牢记你将处理的是你的网站，然后明智的做出选择。

我该怎么添加 Logo，链接，或者是网站图标 (favicon)？

添加网站图标只需将适合的 favicon.ico 添加到域名目录根部即可。暂时还没有简单的添加 Logo 或者是连接的方法，但是在最近的新版本 beta 中会出现。感谢你的耐心！

我能把 Flarum 与 WordPress / Laravel / 其他软件整合么？

那不是很好么！现在它已经是可能的了，但是我们希望会在未来在整合方面做更多的工作。如果你是个开发者，那么敬请参与这一进程吧！

Flarum 会有 [在此插入功能] 么？什么时候？为什么没有？

我们自然希望 Flarum 有数不清的功能和扩展，但是要事第一。在这之前，我们的目标集中在必需品和稳定性上。查看我们的 Roadmap 以关注我们未来将实现哪些功能。

为什么你们还没有修复 [在此插入问题]？

这次的答案仍旧是“要事第一”。如果我们目前还没有修补一个问题（或者分配它为一个 Milestone），那是因为我们正在其他重要的事情上工作。请耐心等待；我们会尽量在稳定版前把它完成。如果你很急的话，欢迎你来参与修复，对本项目做出贡献！##

我能把我的论坛迁移到 Flarum 么？

可以，不过尚需等待。我们现在的重点是让 Flarum 更加稳定，并且功能完备。一旦我们已有将 Flarum 应用于生产环境的信心，我们就会制作导入其他论坛数据的工具，比如 esoTalk，FluxBB，phpBB，等等。

为什么我不能排版我的帖子？

Flarum 目前暂无可视的编辑器——因为我们在 Beta 版之前还没有准备好。请放心，这已被纳入计划，很快就将在核心中出现。

尽管现在还没有可视编辑器，Flarum 已支持 Markdown 和 BBcode 语法。Markdown 是一种用于表示电子邮件和新闻组帖子的，基于结构化文档的纯文本语法格式。如果你对 Markdown 毫不熟悉，请看看这个快速入门以帮助你快速在 Flarum 上排版。

我该怎样将 Flarum 翻译成我的语言？

我们目前还在为语言包制作的过程而工作。我们希望您能在万事俱备前稍加等待。感谢你的耐心！

> 译者：@ttnl Last updated by @ttnl on 17/1/20

使用 Flarum

安装

Beta 版软件

请留意 Flarum 正处在 Beta 版软件阶段。这意味着：

- 它还有很多没完成的功能和 **Bug**，　　而且
- 某些时候 —— 可能早可能晚 —— 它甚至会崩溃！

Beta 阶段关乎解决这些问题并改善 Flarum。我们正在努力使 Flarum 更好，所以我们希望你：

- 不要在生产环境中使用它。如果出错了，我们可帮不了你。况且升级到未来的版本可能是个脏累活儿。
- 有责任感地汇报 **Bug**。书写差劲的 Bug 报告需要花很多时间处理，会扰乱我们添加新功能或者使 Flarum 更稳定的进程。 在你安装前，请阅读我们的贡献规则，你将了解你注册的意义！

系统需求

> 注意：仅就 0.1.0-beta.3 版中，Flarum 使用 Composer 管理依存关系和扩展。这意味着 **Flarum** 没法在没有 **SSH**（命令行）连接的主机上安装。我们未来会保证 Flarum 能用其他的方法安装，使之可被所有人使用。与此同时，如果现在你希望运行 Flarum，你需要寻找允许 SSH 连接的主机。

下为运行 Flarum 所需的一些条件。

- 网页服务器软件： **Apache** (需要支持 mod_rewrite)，**Nginx**，或者**Lighttpd**
- **PHP 5.5+**，需要开启的扩展：mbstring, pdo_mysql, openssl, json, gd, dom, fileinfo
- MySQL 5.5+
- SSH（命令行）连接

安装 Flarum

Flarum 使用 Composer 做为管理依存和扩展的工具。所以，在安装 Flarum 之前，你需要在你的电脑上安装 Composer。然后在 Flarum 应该安装的地方运行此命令：

```
composer create-project flarum/flarum . --stability=beta
```

在此命令运行时，你就可以为你的服务器配置 URL 重写了。最后，使用浏览器访问你的论坛，完成最后的安装步骤。

URL 重写

Apache

Flarum 附带了一个 Apache 的 .htaccess 文件 —— 记得正确地上传它。如果你在使用共享空间，确保你的空间提供商开启了 `mod_rewrite` 。你也可能需要将下述加入到你的 Apache 配置中：

```
<Directory "/path/to/your/forum">
    AllowOverride All
</Directory>
```

Nginx

添加下述配置到你的服务器配置块：

```
location / { try_files $uri $uri/ /index.php?$query_string; }
location /api { try_files $uri $uri/ /api.php?$query_string; }
location /admin { try_files $uri $uri/ /admin.php?$query_string; }

location /flarum {
    deny all;
    return 404;
}

location ~* \.php$ {
    fastcgi_split_path_info ^(.+.php)(/.+)$;
    fastcgi_pass unix:/var/run/php5-fpm.sock;
    include fastcgi_params;
    fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    fastcgi_param HTTP_PROXY ""; # Fix for https://httpoxy.org/ vulnerability
    fastcgi_index index.php;
}

location ~* \.html$ {
    expires -1;
}

location ~* \.(css|js|gif|jpe?g|png)$ {
    expires 1M;
    add_header Pragma public;
    add_header Cache-Control "public, must-revalidate, proxy-revalidate";
}

gzip on;
gzip_http_version 1.1;
gzip_vary on;
gzip_comp_level 6;
gzip_proxied any;
gzip_types application/atom+xml
           application/javascript
           application/json
           application/vnd.ms-fontobject
           application/x-font-ttf
           application/x-web-app-manifest+json
           application/xhtml+xml
           application/xml
           font/opentype
           image/svg+xml
           image/x-icon
           text/css
           text/plain
           text/xml;
gzip_buffers 16 8k;
gzip_disable "MSIE [1-6]\.(?!.*SV1)";
```

Lighttpd

添加下述配置到你的服务器配置块：

```
url.rewrite-if-not-file = (
    "/admin.*" => "/admin.php",
    "/api.*"   => "/api.php",
    "/.*"      => "/index.php"
)
```

故障排除

如果你在安装 Flarum 中遇到问题，记得检查支持论坛的 Installation 标签 —— 也许有人和你遇到了相同的问题！如果没有，发起讨论，我们会尽可能帮你解决。

SMTP

目前还没有配置 SMTP 的图形化界面 (#258)，但您可以手动修改数据库的 `config` 表：

```
mail_driver: smtp
mail_host: ...
mail_port: ...
mail_username: ...
mail_password: ...
mail_encryption: ...
```

导入数据

最终我们希望可以将其他论坛的数据导入到 Flarum 中，这样你就可以放心地迁移到 Flarum 软件。 然而，在这个阶段还为时过早，我们需要先发布稳定版！

> 译者：@justjavac Updated by ttnl 17/1/21

添加语言

在 Flarum 中添加一种新的语言十分容易，只需按照下述的流程下载并安装你选择的语言包即可。

你添加一个语言包之后，你可以把它设置成你的论坛的默认语言。不论何时你感觉不需要其中一个安装过的语言包了，你都可以禁用它。

如果你正在使用任何第三方的扩展，一定要在开始动手之前读这部分。

语言包的安装

第一步，访问 Flarum 社区网站的Extensions > Language标签，并找到你想要安装的语言包。一定要确保下载的ZIP文件是符合你正在运行的 Flarum 版本的。

1. 解压ZIP文件并阅读附上的**readme.txt**文件中的说明。
2. 通过SSH，FTP，或者你的服务提供商的控制面板来登录到你的服务器。
3. 跳转到你的Flarum安装的**extensions/**目录。
4. 创建一个新的子目录。按照说明里指出的那样来命名！
5. 上传解压出的文件夹里的所有东西到你刚刚创建的子目录。
6. 用浏览器跳转到管理界面的**Extensions**页面。
7. 选择你刚刚安装的语言包并启用它。

这就是你要做的所有的事！你现在应该已经能够使用网站的头部的语言选择器，切换你论坛的显示语言至新的语言。

设置默认语言

在你安装了新的语言包并确认它能顺利工作后，你可能想要把它设置为用户的默认显示语言。你可以在管理员面板的 Basic 页面设置这一选项。

禁用语言包

如果你最终决定停止支持一个特定的语言，你可以把它关掉。只需要在管理界面里定位到**Extensions**页面里的语言包并禁用它。

当你的站点只支持一个语言而且你不希望网站的头部出现一个语言选择器，禁用掉一个语言是很有用的。当只启用了一个语言的时候，语言选择器是隐藏的。

第三方扩展

尽管从 Flarum 社区网站上下载的语言包会一般包括所有和 Flarum 捆绑的扩展的翻译，他们不会是一个在你安装的第三方扩展上也起作用的规则。由开发者来决定是否提供并维护他们的扩展的翻译。

所以当你安装一个第三方扩展之前，你应该检查一下来确保它包含有适用于你所安装的所有语言包的翻译。如果你发现一个扩展不支持你需要的一个语言，请直接联系开发者并协商添加必要的翻译。

> 译者：@imcaffrey Updated by ttnl 17/1/21

故障诊断和报告

如果 Flarum 无法安装或者是没有按照预期运行，第一件需要做的事情就是再次检查你的环境是否达到了系统要求。如果你缺失部分 Flarum 的依赖项（例如 PHP 的 `fileinfo` 扩展），你将需要先处理这些问题。 接下来，你应该花上几分钟在支持论坛和问题追踪器内检索。有可能有人已经汇报了这个问题，或者解决方案正在讨论，或者已经有解决方案。在检索过后，如果你仍然没有发现关于这个问题的信息的话，就是需要进行故障诊断的时候了。

第一步：开启调试模式。

在你处理前，你应该开启 Flarum 的调试模式。仅需使用文本编辑器打开 `config.php` 文件，将 `debug` 的值修改为 `true` 。这样，当 Flarum 发生任何错误时，将会显示详细的错误消息。

如果你访问网站时看到一个空白页面，将 PHP 配置文件 **php.ini** 的 `display_errors` 选项设置为 `On` 。 或者，你也可以将以下代码添加到 **flarum/bootstrap.php** 文件的开头：

```
ini_set('display_errors', 'On');
```

当错误修正后，一定要恢复这些选项！

第二步：复现错误。

试着让错误再发生一次。

> Re-translated by @ttnl 17/1/21. First editon translated by @justjavac.

使用 API

每个 Flarum 论坛都有一个公开的 JSON API，可以对论坛数据进行读和写。它符合JSON-API 1.0 规范。

身份验证

API 使用基于令牌的身份验证机制。某些终端不需要身份验证。您可以通过 `/api/token` 取得一个令牌:

```
POST /api/token HTTP/1.1

{
    "identification": "Toby",
    "password": "pass7word"
}

HTTP/1.1 200 OK

{
    "token": "YACub2KLfe8mfmHPcUKtt6t2SMJOGPXnZbqhc3nX",
    "userId": "1"
}
```

然后，您可以在后续的请求头中，包含此身份验证令牌:

```
GET /api/forum HTTP/1.1
Authentication: Token YACub2KLfe8mfmHPcUKtt6t2SMJOGPXnZbqhc3nX
```

论坛

- `GET /api/forum` - 关于论坛的信息，包括 groups 和 tags
- `PATCH /api/forum` - 更新论坛配置

Discussion 讨论

- `GET /api/discussions` - 获取所有的讨论（默认排序规则为 `-time` ）
  - `filter[q]` - 根据 username/gambits 过滤
- `POST /api/discussions` - 创建一个新的讨论

- `GET /api/discussions/:id` - 通过 ID 获取讨论
- `PATCH /api/discussions/:id` - 更新一个讨论
- `DELETE /api/discussions/:id` - 删除一个讨论

Posts 帖子

- `GET /api/posts` - 获取所有帖子
  - `filter[discussion]` - 根据 discussion ID 过滤
  - `filter[user]` - 根据 user ID 过滤
  - `filter[number]` - 根据数字 (discussion 中的位置)
  - `filter[type]` - 根据 post type 过滤
- `POST /api/posts` - 创建一个新的帖子
- `GET /api/posts/:id` - 根据 ID 获取一个帖子
- `PATCH /api/posts/:id` - 更新一个帖子
- `DELETE /api/posts/:id` - 删除一个帖子

Users 用户

- `GET /api/users` - 获取所有用户
  - `filter[q]` - 根据 username/gambits 过滤
- `POST /api/users` - 注册新用户
- `GET /api/users/:idOrUsername` - 根据 ID 和 username 获取一个用户
- `PATCH /api/users/:id` - 更新一个用户
- `DELETE /api/users/:id` - 删除一个用户
- `POST /api/users/:id/avatar` - 更新用户头像
- `DELETE /api/users/:id/avatar` - 删除用户头像

Groups 组

- `GET /api/groups` - 获取所有组
- `POST /api/groups` - 创建一个新的组
- `PATCH /api/groups/:id` - 更新一个组
- `DELETE /api/groups/:id` - 删除一个组

Notifications 通知

- `GET /api/notifications` - 获取所有通知

- `PATCH /api/notifications/:id` - 将一个通知标记为已读

Tags 标签

- `POST /api/tags` - 创建一个新标签
- `PATCH /api/tags/:id` - 更新一个标签
- `DELETE /api/tags/:id` - 删除一个标签

> 译者：@justjavac

构建扩展

简介

Flarum 被设计成轻量级和高扩展的。事实上，Flarum 的大部分特性都是扩展。这种方式使得 Flarum 可定制性很强。用户可以禁用他不需要的特性，或者安装他们想要的扩展。

为了达到这种扩展性，Flarum 拥有丰富的接口和扩展点。文档该部分旨在指导你 Flarum 如何运作，如何使用接口构建自己的扩展。

> 扩展接口还不稳定，接下来几个月内，这种状况会随着我们的不断完善而改变。另外，本文档也未完成。我们需要你的反馈！

核心 vs 扩展

我们怎么界定 Flarum 的核心和扩展？为什么一些特性包含在了核心内而另一些却没有呢？理解这个差异对我们维持 Flarum 生态系统的一致性和高质量很重要。

Flarum 的核心并不意图去包含完整特性，相反，它是作为脚手架，或者说是框架，为扩展提供一个可依赖的构建基础。它只包含对一个论坛来说必要的、基础的、不可缺失的功能，这些功能有讨论、帖子、用户、群组和通知。

绑定的扩展是随 Flarum 一起打包的特性，默认情况下是开启的。这些扩展和其他扩展一样是可以禁用和卸载的。尽管他们并不旨在处理所有使用实例，但这些扩展还是很通用并且被配置得足够好，使得他们能够满足大部分情况。

第三方扩展是其他人制作的、不被 Flarum 团队官方支持的特性。他们应当被构建和用于特定的情形。

如果你想解决核心或者已有的附带扩展的 bug 或者缺陷，我们将很感激你对相应的项目作出贡献而不是分散精力在一个新的第三方扩展上。在Flarum 论坛上新建一个讨论来听听 Flarum 开发者的意见也是个好想法。

Flarum 如何运作

在你构建扩展之前，你一定得先理解 Flarum 如何运作。本节旨在快速浏览组成 Flarum 的各部分。

从顶层来看，Flarum 有 3 层架构：

- 领域**(domain)**层，代表所有 Flarum 数据结构和领域逻辑。
- 接口**(API)**层，以一种标准格式(如 JSON API)暴露这些数据结构。
- 客户**(client)**层，提供默认网络用户来使用接口。

为了构建扩展，你需要了解一点如何去扩展这些层中的每一个。例如锁定(**Sticky**)扩展:

- 扩展领域层，为讨论增加一个新的 `is_sticky` 属性。
- 扩展接口层，允许通过 JSON API 改变属性的值。
- 扩展客户层，增加一个讨论控制条目，该条目被点击时会调用 JSON API。

让我们更深入得了解下每个层，知道他们是如何构建的。

领域

领域层代表着管理论坛数据和为论坛数据建模。这包括用数据库存储数据、提供程序命令去修改数据和控制相关逻辑(例如当一个用户发帖时，我们在用户发帖数上加1)。

领域层使用面向对象的 **PHP**来构建，位于 `Flarum\Core` 这个命名空间里。领域层的每个实体(讨论、帖子等)都有自己的子命名空间。

领域层依赖 Laravel 的数据库组件和 Eloquent ORM。每个实体代表一个 Eloquent 活动记录模型。这些模型对认证、许可、扩展有内建的机制。

为了提供一种修改数据的方式，Flarum 利用命令总线模式以及 Laravel 的总线组件。每个实体都有各种命令与其相关联(例如 `Flarum\Core\Discussions\Commands\StartDiscussion` , `Flarum\Core\Posts\Commands\PostReply`),这些命令都可以被调用来作用在 Flarum 的领域层上。

接口

接口层位于领域层之上，提供能读写论坛数据的公有的 **JSON API**。它符合 JSON-API 1.0 规范

接口层使用面向对象的 **PHP**来构建，存放在 `Flarum\Api` 下。每个接口端都有相对应的操作 **(Action)**(本质上是一个控制器)，这些操作接收请求对象然后返回响应。

大部分操作使用 tobscure/json-api 库来输出一个 JSON-API 文档。来自 Flarum 领域层的数据通过序列化被转换成可使用的格式。写数据操作使用命令总线将命令发送到领域层。

客户

客户层位于接口层之上，提供人们能使用 Flarum 的用户界面。客户层有两部分：一个基础的只读内容的 HTML 版本，和一个交互的单页 JavaScript 应用。两者都使用 JSON API.

客户层 HTML 利用面向对象的 **PHP**，存放在 `Flarum\Forum` 下。和接口层相似，每个论坛的路由（route）有一个对应的操作(本质是控制器)，这个操作能接收 PSR-7 请求并返回响应.

典型的客户端响应包含一个表现页面内容的基础 HTML 样式，和 JavaScript 应用的启动代码。它使用 Laravel 的 视图组件和 Blade 模板系统来构建。

JavaScript 应用基于 Mithril 开发，Mithril 是一个和 React 相似的轻量级渲染框架。JavaScript 应用利用 Gulp 和 Babel 从 ES6 代码库转换而来。

客户层还包含一个 LESS CSS 编译器来为客户端应用编译样式表。

扩展如何运作

现在你已经理解了 Flarum 如何运作，让我们来看看关于如何扩展 Flarum。

事件

在 Flarum 后端的 PHP 代码（例如领域层、接口层、部分客户层）中，有许多事件，而扩展可以监听这些事件，或者对事件作出响应。例如，开始一个讨论后，`DiscussionWasStarted` 事件被触发。扩展可以监听这个事件，这样当一个讨论开始后就能发送某种通知。

在 `Flarum\Events` 命名空间下的所有事件都能被监听.

监听事件很简单，只需要在事件调度程序中调用 `listen()` 方法，并传给它一个事件的类名和控制事件的闭包。你能在打包章节学习到更多关于如何注册事件监听者。

猴子补丁

扩展 Flarum 的前端 JavaScript 应用采用了一种叫猴子补丁的概念.猴子补丁可以在运行时替代一些原有的函数、方法和属性。

Flarum 的前端用户界面有许多组件组成，这些组件都是一个类。扩展可以用猴子补丁的方式修改这些类的方法，适当改变返回值。例如，一个扩展可以猴子补丁 `DiscussionControls` 类的 `moderationControls` 方法来添加一个新的锁定谈论的按钮.

你将会在扩展客户端章节学到更多关于如何在 Flarum 的 JavaScript 应用中运用猴子补丁。

> 译者：@theFool32

打包

配置

在开始写扩展之前，你需要在系统上配置如下几个工具：

- 下载并全局安装 Composer ，为扩展生成自动加载。
- 配置 Node.js 并全局安装 Gulp ，编译扩展的客户端 JavaScript。

除此之外，你也可以安装 Flarum 的 Vagrant 开发者镜像，它帮你自动装好了所有工具。

生成框架

Flarum 有便利的工具来为你生成扩展的框架，这样你就可以马上开始写代码了。打开终端并在论坛根目录运行下面的命令：

```
php flarum/flarum generate:extension
```

工具会向你咨询一些细节。扩展名是扩展独一无二的名称，只能包含数字、字母和连字符(`-`).命名空间用来分离你的代码；你可以把提供者名字加扩展名(例如 `Tobscure\Attachments`)作为命名空间.

一旦你填完所有细节，工具会在你论坛根目录的 `extensions` 目录下创建一个新目录。完成这些可能会花费几分钟的时间。下面让我们看下生成了些什么：

- **bootstrap.php** 如果你的扩展被启用， Flarum 会在每当有请求时加载这个文件。这个文件是扩展启动的地方！你不应该修改它。
- **flarum.json** 包含你扩展的元信息(标题、描述、作者、依赖项)。更多信息请往下看。
- **js/** 存放扩展的 JavaScript 文件。你将在扩展客户端了解到更多。
- **less/** 存放扩展的 CSS 文件。你将在主题了解到更多。
- **locale/** 存放扩展的翻译文件。你将在国际化了解到更多。
- **migrations/** 当扩展被安装/升级/卸载时会运行数据迁移。你将在域扩展了解到更多。
- **src/** 存放扩展的后端源代码，将被使用 PSR-4 标准的 Composer 自动加载。

flarum.json

文档待写。

事件订阅者

就像你从介绍里了解到的一样，扩展工作的唯一方式就是监听和响应事件。让我们看下这是如何实现的。

我们将从头开始。如果你看过 `bootstrap.php` ，你会发现它返回一个扩展类的名字。如果扩展启用了，在每个请求的开始，Flarum 会包含这个文件并且把这个类注册为一个服务提供者。这个类是你处理事务的关键。

我们再继续看，Flarum 在 `src` 目录下生成了这个类。如果你仔细看，你会发现这个独立的方法:

```
public function listen(Dispatcher $events)
{
    $events->subscribe('Your\Namespace\Listeners\AddClientAssets');
}
```

当事件系统建立起来后，Flarum 就会调用每个扩展服务提供者的 `listen` 方法。它传递一个 Laravel 的消息分发实体，因此你能如你所愿得监听事件。

但我们在这做了些不同的事。我们调用 `subscribe` 方法去注册一个事件订阅者而不是直接添加事件监听器. 订阅者是分组相关事件监听器的一种，它能帮助你组织代码。你可以按你喜欢的方式组织你的事件监听器，但 Flarum 的附带扩展采用订阅者的任务 + 订阅者名(例如 `AddClientAssets` 或 `AddApiAttributes`)的约定。

好了，让我们再看下生成的 `AddClientAssets` 监听器。

```
use Flarum\Events\RegisterLocales;
use Flarum\Events\BuildClientView;
use Illuminate\Contracts\Events\Dispatcher;

class AddClientAssets
{
    public function subscribe(Dispatcher $events)
    {
        $events->listen(RegisterLocales::class, [$this, 'addLocale']);
        $events->listen(BuildClientView::class, [$this, 'addAssets']);
    }
```

这是我们注册事件监听器的地方。我们导入 event 类，然后在订阅者的 `subscribe` 方法里调用分发器的 `listen()` ，传递的参数是我们希望订阅的事件名称和处理事件的方法。

这些特定的事件监听器用来处理默认的本地化语言包、JavaScript 和 CSS 文件，这些都是在创建扩展的时候由系统自动生成。你可以在扩展客户端章节了解到关于他们得更多信息。

> 译者：@theFool32

扩展域

如介绍所述，域(Domain)层负责管理和抽象化论坛数据。这包括在数据库中贮存数据，提供编程命令以修改数据，并处理相关逻辑（比如，当用户发了帖子，我们就在用户的帖子数统计上加一）。

在本节我们将深入探讨如何扩展 Flarum 的域。

迁移

如果您的扩展引入了一种新的实体 (如标签)，或者将新属性添加到了一个现有的实体 (例如 `discussions` 的 `is_sticky` 属性)，那么你需要更新 Flarum 的数据库架构，来存储新的数据。

可通过迁移（**migrations**）来实现这一工作；Flarum 中的迁移基于 Laravel 的实现。迁移文件必须放在扩展的 `migrations` 文件夹下，命名时应包括时间戳和对其所作修改的描述，如下所示：

```
2015_02_24_000000_create_tags_table.php
```

此文件应该包括一个扩展自 `Flarum\Migrations\Migration` 的类，并附有相同的描述。每个迁移类包括两个方法：`up` 和 `down` 。`up` 方法在你的扩展启用时调用（如果该迁移以前没有进行过），因此你可以在数据库中添加新的表、列、索引等等，亦可执行其他安装操作。`down` 操作在扩展卸载时被调用，应能够回滚 `up` 操作所执行的行为。

迁移可调用 `$schema` 变量，此为 Laravel 的架构生成器的一个实例，用来对数据库架构进行调整：

```
use Illuminate\Database\Schema\Blueprint;
use Flarum\Migrations\Migration;

class CreateTagsTable extends Migration
{
    public function up()
    {
        $this->schema->create('tags', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name', 100);
            $table->string('slug', 100);
            $table->text('description')->nullable();
        });
    }

    public function down()
    {
        $this->schema->drop('tags');
    }
}
```

如果需要在不重新启用某个扩展的情况下运行迁移，只需简单地执行升级命令即可：

```
php flarum upgrade
```

扩展模型

为了使用您设置的新数据库结构，你需要创建新的模型(Model)或对现有的模型加以扩展。在 Flarum 中，每个模型都扩展自 `Flarum\Core\Model` ，此亦为 Laravel 的 `Eloquent\Model` 类的扩展。在此详细了解 Eloquent 模型。

属性

在 Eloquent 模型中，你无需为定义属性而烦恼。你可以像调用实例的属性一样调用它，比如 `$discussion->is_sticky` 。这一切使得事情变得很简单。

然而有一个例外：如果你添加了一个日期属性，你需要告诉模型此属性应被视为日期。可通过侦听 `ModelDates` 事件来实现：

```
use Flarum\Core\Discussions\Discussion;
use Flarum\Events\ModelDates;

$events->listen(ModelDates::class, function (ModelDates $event) {
    if ($event->model instanceof Discussion) {
        $event->dates[] = 'stickied_at'; // stickied_at 被作为日期属性
    }
});
```

验证

每次保存模型时，其数据将根据使用 Laravel 验证组件以一套规则加以验证。扩展可以修改这些规则，所以您可以在模型属性上执行额外的验证。

```
use Flarum\Core\Users\User;
use Flarum\Events\ModelValidator;

$events->listen(ModelValidator::class, function (ModelValidator $event) {
    if ($event->model instanceof User) {
        // Make usernames require at least 5 characters
        // username 至少 5 个字符
        $event->validator->mergeRules('username', 'min:5');
    }
});
```

关系

如果您需要对已有的模型定义一个新的关系，可以使用 `ModelRelationship` 事件。每个未知的方法或属性连接模型时，都将调用此事件。如果你关系的名称与方法的名称匹配，你应该使之返回一个 Eloquent Relation 对象，该对象应使用模型 `hasOne` , `belongsTo` , `belongsToMany` , 或者 其他关系方法。

```
use Flarum\Core\Discussions\Discussion;
use Flarum\Events\ModelRelationship;

$events->listen(ModelRelationship::class, function (ModelRelationship $event) {
    if ($event->model instanceof Discussion && $event->relationship === 'category') {
        return $event->model->belongsTo('category');
    }
});
```

数据持久化

如介绍中所述，Flarum 使用命令总线模式（译者注：请参见本文档。）提供对数据进行修改的方法。为了能够使一个核心实体中的所有新属性/关系保存到数据库，你需要注意会发生什么。

让我们用一个例子来说明这个过程。当我们对 `/discussions/123` 入口点进行 PATCH 请求时，将会进行下列过程：

1. 请求传递到合适的 API 行为类 (`Flarum\Api\Actions\Discussions\UpdateAction`)。
2. 该操作创建 `Flarum\Core\Discussions\Commands\EditDiscussion` 命令的新实例，并把请求输入进行封装。
3. 该行为把命令发送至命令总线，将其传递给适当的命令处理程序: `Flarum\Core\Discussions\Commands\EditDiscussionHandler`
4. `EditDiscussionHandler` 根据输入的命令，对讨论模型进行修改，并保存此讨论。

在第四步时 `DiscussionWillBeSaved` 事件会被发送。这是在保存前，扩展用来检查输入、并对模型进行的附加修改的时机。

```
use Flarum\Events\DiscussionWillBeSaved;

$events->listen(DiscussionWillBeSaved::class, function (DiscussionWillBeSaved $event)
{
    if (isset($event->data['attributes']['isSticky'])) {
        $event->discussion->is_sticky = $event->data['attributes']['isSticky'];
    }
});
```

权限

Flarum 包括一个功能强大的权限系统。它使得扩展能自行定义操作逻辑，允许或不允许某些用户或组对某些实体执行特定行为。

模型锁

当你需要确定是否用户可以在模型上执行操作时，可调用该模型的 `can()` 方法 (在 `Flarum\Core\Support\Locked` 中定义) 。当用户被允许就返回 `ture` ，反之则为 `false` 。

```
$allowed = $discussion->can($user, 'sticky');
```

扩展可以使用 `ModelAllow` 事件挂接于此，返回 `ture` 或 `false` 来授予或拒绝用户相关权限。

```php
use Flarum\Core\Discussions\Discussion;
use Flarum\Events\ModelAllow;

$events->listen(ModelAllow::class, function (ModelAllow $event) {
    if ($event->model instanceof Discussion && $event->action === 'sticky') {
        // Allow the user to sticky the discussion if they authored it
        // 允许作者置顶自己的讨论
        if ($event->actor->id === $event->model->user_id) {
            return true;
        }
    }
});
```

优先级

值得注意的是，当注册一个事件监听器时，你可以传递一个整数作为第三参数，从而给事件监听器对应的优先级。这对于 `ModelAllow` 事件相当有用，能使在返回第一个非空值时，停止执行后续的事件监听器。

组的权限

Flarum 具有关于组和对应授予的基本权限的映射（可在后台权限页加以修改）。此信息存储在数据库的权限表中，并可以在用户模型上使用 `hasPermission()` 方法访问:

```php
$granted = $user->hasPermission('discussion.sticky');
```

对管理员组的用户而言，此方法总是返回 `true`（也就是说，管理员有权进行一切事务）。其他情况，它将返回 `true` 或 `false`，具体取决于用户所在的组和对应权限映射。

这个系统与模块锁系统相连接，从而使得在被授权的某组中的用户能做规定的行为。默认情况下，在 `discussion` 模块上检查用户是否能做对应行为时，会检查其所在组的 `discussion.{action}` 许可。对于 `posts`、`users` 和 `groups` 上亦如此。

若要了解如何在后台中添加权限设置，请参阅 管理 章节。

可见范围

模块锁仅在检查已取得数据库的模型时有用。为了筛选哪种模式应该最先从数据库中取出 (比如确定哪些讨论/帖子是对用户可见的)，Flarum 使用一种机制称为可见范围。

可见范围只是一套能在查询数据库模型时可应用的条件语句。可以使用 `whereVisibleTo()` 方法应用此范围（该方法在 `Flarum\Core\Support\VisibleScope` 中定义）：

```
// Olny get discussions which are visible to $user
// 仅获取对 $user 用户可见的帖子
$discussions = Discussion::whereVisibleTo($user)->get();
```

扩展可以使用 `ScopeModelVisibility` 事件挂接于此，应用限定条件于查询生成器：

```
use Flarum\Core\Discussions\Discussion;
use Flarum\Events\ScopeModelVisibility;

$events->listen(ScopeModelVisibility::class, function (ScopeModelVisibility $event) {
    if ($event->model instanceof Discussion) {
        // Only let the user see discussions which they started
        // 用户只能看到自己发起的讨论
        $event->query->where('start_user_id', $event->actor->id);
    }
});
```

帖子可见范围

我们使用一个简单的模式来筛选用户可见帖子：

```
$posts = $discussion->postsVisibleTo($user)->get();
```

同样地，扩展可使用 `ScopePostVisibility` 事件来挂接到此方法，并对查询生成器应用限定条件。此事件包含了对帖子讨论的查询。

> 译者：@ttnl

API

正如介绍所述，Flarum 有一个公开的 **JSON API** 允许你读写论坛数据。它符合 JSON-API 1.0 规范。

在本节中我们将深入探讨如何扩展 Flarum 的 API。

JSON-API 和序列化程序（Serializers)

Flarum 使用 tobscure/json-api 库来输出符合 JSON-API 规范的格式化数据。这涉及序列化程序的使用，该程序是 `Flarum\Api\Serializers` 命名空间下的一个类，能将 Flarum 的数据转换成一个可被公开接受的格式。

属性

可使用 `ApiAttributes` 事件对序列化程序进行增改：

```
use Flarum\Api\Serializers\DiscussionSerializer;
use Flarum\Events\ApiAttributes;

$events->listen(ApiAttributes::class, function (ApiAttributes $event) {
    if ($event->serializer instanceof DiscussionSerializer) {
        $event->attributes['isSticky'] = (bool) $event->model->is_sticky;
        $event->attributes['canSticky'] = $event->model->can($event->actor, 'sticky');
    }
});
```

关系

可使用 `ApiRelationship` 事件来添加新的关系模型。Flarum 的底层 Serializer 类提供了两个方便的方法(`hasOne` 和 `hasMany`)，可构建一个 `Tobscure\JsonApi\Relationship` 对象，用来处理模型之间的关系。

```
use Flarum\Api\Serializers\DiscussionSerializer;
use Flarum\Events\ApiRelationship;

$events->listen(ApiRelationship::class, function (ApiRelationship $event) {
    if ($event->serializer instanceof DiscussionSerializer && $event->relationship ===
'tags') {
        return $event->serializer->hasMany('Flarum\Tags\Api\TagSerializer');
    }
});
```

为使这些关系能被对应文档所链接或包含，你必须把关系与适当的行为加以联系 —— 阅读下面。

行为

Flarum 的 API 端点(endpoint) 在 `Flarum\Api\ApiServiceProvider` 中定义。通过路由（route），每个端点都与一个对应的行为（**Action**）类相连接，该类实际上是一个控制器。这个行为类可以接受一个 `Flarum\Api\Request` 对象，并以 PSR-7 格式响应。

在 `Flarum\Api\Action` 命名空间下有大量的抽象行为类，可以扩展这些类来方便地实现数据库操作（译注：原文作 CRUD actions, 请参见维基百科）。其中包括 `SerializeResourceAction` 和 `SerializeCollectionAction` 类，这两个类将为你进行设置和在序列化程序中传递数据的所有工作。初学者仅需实现 `data` 方法并以 Eloquent 模型（译注：此为 Laravel 功能，请参见官方文档） 或集合(Collection)形式返回即可。

更改行为属性

`data` 方法接受一个请求对象和几个额外的参数：`sort` ，`include` ，`link` ，`limit` 和 `offset` 。这些参数使用行为类中的配置来进行处理，扩展程序可以使用 `BuildApiAction` 事件修改这些配置，以允许添加新的关系、新的可排序字段，诸如此类。

```
use Flarum\Api\Actions\Discussions;
use Flarum\Events\BuildApiAction;

$events->listen(BuildApiAction::class, function (BuildApiAction $event) {
    if ($event->action instanceof Discussions\IndexAction ||
        $event->action instanceof Discussions\ShowAction ||
        $event->action instanceof Discussions\CreateAction ||
        $event->action instanceof Discussions\UpdateAction) {
        // Include the tags relationship on discussion resources by default
        $event->addInclude('tags');
    }
});
```

添加 API 端点

如果您需要添加一个新的 API 端点，可从任何 Action 基类加以扩展，然后使用 `RegisterApiRoutes` 事件注册新路由即可：

```
use Flarum\Events\RegisterApiRoutes;

$events->listen(RegisterApiRoutes::class, function (RegisterApiRoutes $event) {
    $event->get(
        '/tags', // The endpoint URI
        'api.tags.index', // A unique name for the endpoint
        'Flarum\Tags\Api\IndexAction' // The Action class to handle the request
    );
});
```

> 译者：@ttnl

客户端

在简介一节中我们提到，客户端层暴露了一个供人类使用的用户界面。 客户端由两部分组成：一个只读的简单 HTML 页面和一个可交互的单页 JavaScript 应用。两者都从 JSON API 读取数据。

路由

论坛所有的默认路由都在 `Flarum\Forum\ForumServiceProvider` 中注册。每个路由有一个对应的 **Action** （动作）类， 这个类本质上就是控制器。每个动作接受一个 PSR-7 请求，并返回一个响应。

`Flarum\Forum\Actions\ClientAction` 是动作类的基类，它会构建一个 `ClientView` 实例，并将这个实例回应给客户端。 `ClientView` 是一个高度可定制的、包括前端 JavaScript 应用所需要的所有资源文件的主模板。例如你可以：

- 用 `setTitle()` 设置文档标题。
- 用 `setContent()` 设置页面的 SEO 内容（在 `<noscript>` 标签中显示的内容）。
- 用 `setDocument()` 设置页面中需要预加载的 API 响应。
- 用 `addHeadString()` 或 `addFootString()` 在页面的头部或尾部添加 HTML 内容。

新的论坛路由可以通过 `RegisterForumRoutes` 事件来注册。例如：

```
use Flarum\Events\RegisterForumRoutes;

$events->listen(RegisterForumRoutes::class, function (RegisterForumRoutes $event) {
    $event->get(
        '/tags', // 路由 URI
        'forum.tags', // 路由的唯一名字
        'Flarum\Tags\Forum\TagsAction' // 响应请求的动作类（可选）
    );
});
```

定义响应请求的动作类（第三个参数）是可选的。如果没有指定，Flarum 会用默认的空白 `ClientAction` 来响应。

JavaScript 应用

Flarum 最好的一部分就是运行飞快的交互式 JavaScript 前端应用。Flarum 的前端采用一个和 React 很像的轻量级的渲染框架 Mithril 构建。 下面的内容假设你对 Mithrill 很熟悉，并且了解一些 React 的思想。

编译资源文件

Flarum 的前端 JavaScript 应用采用 ES6 编写， 并通过 Gulp 与 Babel 转译成 ES5。 前端大量使用 ES6 模块， 并转译成 System.js 格式。扩展的 JavaScript 也应当如此编写。

Flarum 为你生成的扩展框架包含了所有制作扩展所需的必要工具。 切换到 `js/forum` 目录，执行下面的命令就可以编译你扩展的 JavaScript 文件。

```
gulp watch
```

编译后的 JavaScript 文件会输出到 `js/forum/dist/extension.js` 。这个文件必须注册到客户端的资源管理器中，否则不会加载。 另外，还需要注册一个加载器，这样在前端 JS 应用启动的时候才会执行你的扩展代码。 你可以在 `BuildClientView` 事件中这么做（自动生成的框架中已经包含了这部分代码）：

```
use Flarum\Events\BuildClientView;

$events->listen(BuildClientView::class, function (BuildClientView $event) {
    $event->forumAssets([
        __DIR__.'/../../js/forum/dist/extension.js'
    ]);

    $event->forumBootstrapper('extension-name/main');
});
```

运行时扩展（**Monkey Patching**）

在加载器中注册了你的 `main` 模块后，前端应用会在开始执行真正的逻辑前运行你在 `js/forum/src/main.js` 中写的代码。 这样，你的扩展就有机会修改你想要改的东西了。

在简介一节中我们曾一笔带过的提到我们可以通过 monkey-patching 替换函数。Flarum 提供了两个帮助你操作的函数。 这两个函数是 `extend` 和 `override` ，可以通过 `flarum/extend` 导入。调用这两个函数时，只需传入你想要修改的对象（通常来说是某个类的原型）、你想要修改的属性和一个回调函数即可。

`extend` 函数中，回调函数是在原来的方法 运行完之后 被调用的。它会把原来的执行结果传入你的回调函数，这样你可以修改返回值。

`override` 函数中，回调函数会 替换原来的方法。它会将原来的方法传入，如果你需要，可以在回调中随时调用它。

下面的这两个例子演示了这两个函数的使用方式：

```
import {extend, override} from 'flarum/extend';

class Foo {
  aMethod() {
    return {
      color: 'red'
    };
  }

  bMethod() {
    return 'Gday';
  }
}

const foo = new Foo();
foo.aMethod(); // {color: 'red'}
foo.bMethod(); // 'Gday'

extend(Foo.prototype, 'aMethod', value => {
  value.color = 'yellow';
});

override(Foo.prototype, 'bMethod', original => {
  return original() + ' mate';
});

foo.aMethod(); // {color: 'yellow'}
foo.bMethod(); // 'Gday mate'
```

在运行时扩展方法的时候，你需要确认你扩展或者替换的方法和原来的方法签名（参数的数量和类型、返回值的类型）一致。

组件

组件是一些构建 Fluarm 的 UI 的可重用片段。你可以在 这里 找到所有的组件。

组件是在 Mithril 的组件系统 上用 React 的方式实现的。 一个组件必须继承 `flarum/Component` 并实现 `view()` 方法（默认的扩展工具会将 JSX 转义成 Mithril 的 `m()` 函数）。

```
class WelcomeMessage extends Component {
  view() {
    return (
      <div className="WelcomeMessage">
        <h1>Hello, {this.props.name}!</h1>
      </div>
    );
  }
}
```

一个组件可以通过 `config()` 方法和它的根 DOM 元素互动。这个方法和 Mithril 的 `config` API 几乎是一样的实现，只有一点不同：它没有传入第一个参数（`element`）。你可以用 `$()` 来操作这个元素，就像用一个 jQuery 对象一样。

```
config(isInitialized, context) {
  if (isInitiailized) return;

  this.$().on('click', () => alert('clicked'));
}
```

对象列表（ItemList）

`ItemList` 类是一个通过名字来收集和整理对象的数组。例如：

```
import ItemList from 'flarum/utils/ItemList';

const items = new ItemList();

items.add('foo', 'Foo');
items.add('bar', 'Bar');
items.add('qux', 'Qux', 10); // 更高优先级的对象会排在前面

delete items.foo; // 删除一个对象

items.bar.content = 'Baz'; // 改变一个对象的内容

items.toArray(); // ['Qux', 'Baz']
```

许多组件和其它的实用工具会返回一个 `ItemList` 的实例（通常来说它们的方法名字以 Items 结尾）。例如，`CommentPost` 组件的 `headerItems()` 方法就返回一个 `ItemList` 的实例，其中包含每条帖子的头部。你可以用运行时扩展来修改它们：

```
import CommentPost from 'flarum/components/CommentPost';

extend(CommentPost.prototype, 'headerItems', function(items) {
  items.add('location', '这个用户住在' + this.props.post.user().location());
});
```

应用全局变量

`app` 全局变量是 `flarum/ForumApp` 的实例，你可以通过这个变量来访问整个应用级别的组件实例和实用工具。

存储（**Store**）

存储（Store） 是一个和 Flarum 后端 API 交互的对象，可以通过 `app.store` 来访问。

存储会将在 API 响应中的资源和 Models 关联起来。Models 会让访问对象和关系变得容易起来。 例如，主题的 model 的一部分代码是这样的：

```
class Discussion extends mixin(Model, {
  title: Model.attribute('title'),
  startTime: Model.attribute('startTime', Model.transformDate),
  startUser: Model.hasOne('startUser'),
  posts: Model.hasMany('posts'),
```

要通过 API 获得一个主题的 model，用 store 的 `find` 方法就行了。`find` 方法会返回一个 promise。例如：

```
app.store.find('discussions', 1).then(discussion => {
  discussion.id(); // "1"
  discussion.title(); // "Discussion title"
  discussion.startUser(); // flarum/models/User 对象
});
```

你也可以用 `find` 方法进行更加复杂的查询，例如：

```
app.store.find('discussions', {
  sort: '+time',
  page: {limit: 5},
  filter: {q: 'test'}
});
```

`find` 方法会无条件地发起一个 API 请求。如果你想要从 store 的本地缓存中获取 model 避免发起 API 请求， 那就得用 `getById` 和 `all` 这两个方法了。例如：

```
const discussion = app.store.getById('discussions', 1);
const discussions = app.store.all('discussions');
```

Model 的属性和关系可以用 `save` 来存储到服务器，也可以用 `delete` 来删除。这两个方法都会返回一个 promise。例如：

```
discussion.save({
  title: '新标题',
  relationships: {
    tags: [] // Tag model 的数组
  }
});
discussion.delete();
```

新的数据库记录也可以用 `createRecord` 方法来创建。例如：

```
const discussion = app.store.createRecord('discussions', {
  title: '新的主题'
});
discussion.save();
```

路由

你可以通过 `app.routes` 来添加或修改客户端的路由。路由对象的键名是一个唯一的名字，而值则是有 `path` 和 `component` 两个属性的对象。注意，修改路由只能在初始化的时候完成。

```
app.routes.tag = {path: '/tag/{slug}', component: TagPage.component()};
```

你可以用 `app.route()` 来生成一个路由对应的 URL。例如：

```
<a href={app.route('tag', {slug: 'blog'})} config={m.route}>博客</a>
```

> 译者：@oott123

主题

Flarum 把 LESS 文件编译成 CSS 文件，然后担任客户端的一部分，关于论坛的定制很容易，在管理员面板外观部分可以通过点击“编辑自定义 CSS”添加你自己的 LESS/CSS。

编译资源

扩展框架包含一个空的 `LESS/extension.less` 文件，你可以把任何自定义的 LESS/CSS 样式写入这个文件。这个文件使用的是 `BuildClientView` 事件注册：

```
use Flarum\Events\BuildClientView;

$events->listen(BuildClientView::class, function (BuildClientView $event) {
    $event->forumAssets([
        __DIR__.'/../../less/extension.less'
    ]);
});
```

变量

系统定义了许多 LESS 变量，包括定义主题颜色和其他。它们被列在 variables.less。当你为扩展实现配色方案时，你应该使用这些变量。

规范

Flarum 使用 BEM 风格命名 CSS 类。

```
.ComponentName-child--modifier
```

该组件名称组成部分应该与存在的 JavaScript 组件的名称保持一致。

模板

如果需要，您可以使用自定义模板更改论坛布局。(默认的位于这里。）您的自定义模板必须包含所有相同的ID元素。

```
use Flarum\Events\BuildClientView;
use Flarum\Forum\Actions\ClientAction as ForumClientAction;

$events->listen(BuildClientView::class, function (BuildClientView $event) {
    if ($event->action instanceof ForumClientAction) {
        $event->view->setLayout(__DIR__.'/../forum.blade.php');
    }
});
```

> 译者：@Seevil

国际化

This page is under construction.

分发

分发的目标是使别人能轻松的安装扩展，只需把扩展放到 extensions 文件夹里即可。你不能要求他们安装某种工具或者编译什么代码——那是你作为一个开发者应该完成的工作！

所以在你打包扩展之前，需要安装好所有的 Compser 依赖项，并且编译你的 JavaScript 程序。

一个打包脚本的示例：

```
#!/usr/bin/env bash

base=${PWD}

# Unzip an archive of the latest committed code
# 解压 Flarum 最终版
rm -rf /tmp/extension-release
mkdir /tmp/extension-release
git archive --format zip --worktree-attributes HEAD > /tmp/extension-release/release.z
ip
cd /tmp/extension-release
unzip release.zip -d ./
rm release.zip

# Install all Composer dependencies
# 安装 Composer 依赖
cd /tmp/extension-release/flarum
composer install --prefer-dist --optimize-autoloader --ignore-platform-reqs --no-dev

# Compile JavaScript
# 编译 JavaScript
# Assumes: npm install -g gulp flarum-gulp babel-core
cd /tmp/extension-release/js
for app in forum admin; do
  cd "/tmp/extension-release/js/${app}"
  npm link gulp flarum-gulp babel-core
  gulp --production
  rm -rf "/tmp/extension-release/js/${app}/node_modules"
done
```

> 译者：@ttnl

管理

- Adding assets to the admin client
- Adding a nav item/page
- Setting extension settings

格式化

- s9e/TextFormatter
- Formatting events

通知

- How the notifications system works - blueprints
- Registering on the backend
- Registering notification components in the client
- Adding notification preferences to the grid

帖子

搜索

- Gambits

Translating Flarum

Introduction

This page is under construction.

Packaging

This page is under construction.

本地化

翻译

扩展框架包括一个空的 `locale/en.yml` 文件，你可以把你的英语翻译放在这里。键名需要使用 `snake_case` 。 他们应该放在和扩展名相同的 namespace 下。翻译可以包含 `{placeholders}` （占位符）。

（译者注： `snake_case` 是一种常见的命名法，他使用下划线来分割每个单词，比如 `register_locales` ，通常用在 C 语言或者 javascript 中。对应的还有一种 CamelCase，顾名思义就是驼峰命名法，一般使用在 Java 中，比如 `RegisterLocales` ）

```
tags:
  tag_new_discussion_title: Choose Tags for Your Discussion
  edit_discussion_tags_title: "Edit Tags for {title}"
```

你可以使用 Flarum 的本地管理器（locale manager）的 `RegisterLocales` 事件注册这些语言包。扩展框架默认注册以下语言包：

```
use Flarum\Events\RegisterLocales;

$events->listen(RegisterLocales::class, function (RegisterLocales $event) {
    $event->addTranslations('en', __DIR__.'/../../locale/en.yml');
});
```

为了使语言包能被编译成 JavaScript 文件，以便让 JavaScript 客户端调用，它们必须显式地通过 `BuildClientView` 事件注册。 如果你指定了一个键名，它所有的子元素也会被包括进来。

```
use Flarum\Events\BuildClientView;

$events->listen(BuildClientView::class, function (BuildClientView $event) {
    $event->forumTranslations([
        'tags.tag_new_discussion_title',
        'tags.edit_discussion_tags_title'
    ]);
});
```

你需要从网站的 `assets` 目录里删除 `forum-en-xxxx.js` 文件，可以强制性重新编译本地的 JavaScript。

翻译会调用客户端的 `app.trans()` 方法：

```
app.trans('tags.edit_discussion_tags_title', {title: discussion.title()});
```

复式规则

如果一个 `count` 参数传递进来时，翻译器将寻找子节点的多个规则。默认的复式规则是英语：如果计数为 1，则查找 `one` 键，否则查找 `other` 键：

```
tags:
  choose_primary_tags:
    one: Choose a primary tag
    other: "Choose {count} primary tags"
```

```
app.trans('tags.choose_primary_tags', {count: 1}); // 'Choose a primary tag'
app.trans('tags.choose_primary_tags', {count: 7}); // 'Choose 7 primary tags'
```

配置

您可以配置复式翻译系统的规则或者其他翻译器选项：通过给本地化管理器（locale manager）注册一个 JavaScript 配置文件：

```
// locale/en.js
app.translator.plural = function(count) {
    return count === 1 ? 'one' : 'other';
};

// 你也可以做一些其它的事，比如：配置 moment.js 的本地化文件
```

```
use Flarum\Events\RegisterLocales;

$events->listen(RegisterLocales::class, function (RegisterLocales $event) {
    $event->addJsFile('en', __DIR__.'/../../locale/en.js');
});
```

Flarum 国际化语言包 https://github.com/justjavac/flarum-i18n-zh

> 译者：@justjavac

Updating

This page is under construction.

Distribution

This page is under construction.